Advanced Software Fault Tolerance Strategies For Mission Critical Spacecraft Applications

for
NASA Ames Research Center/IV&V Facility
Software Initiative UPN 323-08

Task 1 Interim Report April 30, 1999
Identify Potential FSW Design and Development Methodologies

Prepared by JSC NX Technology Division

Approved by M. Himel/Chief, NX Technology Division

Purpose and Scope

The purpose of this interim report is to provide a survey of empirically validated techniques for providing software fault tolerant systems or extremely reliable, i.e. near error free, software systems. A secondary objective is to determine if any conceptually new approaches had been developed or validated since the appearance of the originally fault tolerance approaches: N-version programming and recovery.

System design to accommodate hardware failures, whether in the primary computer or in the sensors/effectors is well understood, albeit somewhat less than perfectly practiced. The real challenge facing a designer today is to ensure correct or, in the crewed spacecraft arena, survivable, system performance when executing on less than perfect software. These flaws range from invalid, incomplete, or inconsistent system specifications, to incorrect design and implementation. The subject of this report is limited to software fault tolerance, i.e. methods that will lead to correct system behavior even in the face of software errors.

An extensive literature search to identify methods, tools, and techniques claimed to provide either highly reliable or fault tolerant software systems provides some insight into the current state of the art. Sources consulted are listed in Appendix A. The methods, tools, and techniques identified and to be discussed are:

- Methods (also called methodology) - Conceptual ideology for the development of a fault tolerant system
    - N-version
    - Recovery Block
    - Distributed Recovery Block
    - Formal Methods
- Tools - Items which contribute to the development, implementation, or testing of a fault tolerant system

  Products - Tools which may be purchased and implemented to provide support for a fault tolerant system
    - ReSoFT - Reusable Software Fault Tolerance
    - CORBA - Common Object Request Broker Architecture
    - Spin - a software package that supports the formal verification of distributed systems
    - χSUDS - Software Understanding & Diagnosis System

  Aids - Tools used to identify critical functions of the system for incorporation into a fault tolerant system
    - FMEA
    - Fault Tree Analysis
- Techniques - Structures or concepts used to implement the above methods
    - Assertions
    - Try Blocks/Retry Blocks
    - Acceptance Tests
    - Scenario based testing
    - Voting
    - Heartbeat
    - Watchdog
    - Checkpointing
    - Data Re-expression
    - State Model Checking
- Ad Hoc Techniques - Techniques that make use of special knowledge about the system or its environment and generally can’t be extended to other systems.
    - System consistency checks

- Statistical analysis of inputs
- graceful degradation

Findings

1. N-version and recovery still form the basis of all software fault tolerant systems.
2. Formal methods show great promise but are hampered by several drawbacks:
    - Knowledge required is usually not part of the computer science curriculum,
    - Unable to identify incomplete requirements,
    - Very expensive to fully implement
3. The use of assertions can be quite powerful and is within the grasp of almost all developers.
4. Model state checking is a strong approach to identifying incomplete requirements.
5. The real world will require a combination of approaches, for example:
   The distributed recovery block method is designed around the use of try/retry blocks, acceptance tests, voter mechanisms, and object oriented coding. Given these elements, it would seem logical that critical functions could be identified (using fault trees etc.) for incorporation into objects which would be independently developed (as in n-version) and run on different processors. Voting mechanisms and acceptance tests could then be used to validate the outputs and determine which data to act on. Non-critical functions could be handled in a conventional manner. This grants the benefits of n-version for the most critical functions and limits the costs.

Discussion of SW development methods, tools, techniques

Methods - Conceptual ideology for the development of a fault tolerant system

N-version - multiple versions of software are developed which perform the same function and whose output is compared.

The usual approach is to separately implement multiple versions of the same requirements set. It often happens that the development teams are not quite as independent as imagined because of training or education and the same generic design error will appear in the various versions.

The space shuttle's on-board GNC software uses an extreme N-version approach, although it is seldom referred to as such. The two versions are known as the Primary Avionics Software System (PASS) and the Backup Flight Software System (BFS). The two versions are developed by two separate programming teams from different requirements. The fundamental architectural approaches are different by design. PASS makes use of an asynchronous interrupt driven architecture, modified by the need to communicate with BFS, while BFS uses a time-slice synchronous approach.

Recovery Block - capture the process inputs so that when a fault occurs the system can recover the data, reprocess, and continue operation

Distributed Recovery Block - Fundamentally, an object oriented recovery block scheme run across multiple processors.

Formal Methods - axiomatic methods that rigorously demonstrate, using first order predicate logic, the correctness of the code, i.e. that the code implements the requirements perfectly.

Formal methods have a long history of showing great promise. They can, in theory, automatically verify the system's implementation, as well as identify incomplete or inconsistent requirements sets. The primary drawbacks are the amount of work that goes into proving anything other than a trivial system and the lack of knowledge among practicing developers of the methods of first order predicate logic. While much progress has been made in the realm of automatic proofs a complete formal approach to any system development task remains rare.

Tools - Items which contribute to the development, implementation, or testing of a fault tolerant system

Products - Tools which may be purchased and implemented to provide support for a fault tolerant system

ReSoFT - Reusable Software Fault Tolerance - An integrated (hw/sw) environment for a fault tolerance testbed. ReSoFT is primarily designed for experimentation, through fault-injection, with various software fault tolerant approaches. ReSoFT is developed by SoHaR Incorported.

CORBA - Common Object Request Broker Architecture - Standard specification for the development of distributed object-oriented applications. CORBA doesn't support real-time implementations.

SPIN - Spin is a software package that supports the formal verification of distributed systems. The software was developed at Bell Labs in the formal methods and verification group. Spin can be used as a full linear temporal logic (LTL) model checking system, supporting all correctness requirements expressible in linear time temporal logic, but it can also be used as an efficient on-the-fly verifier for more basic safety and liveness properties. Many of the latter properties can be expressed, and verified, without the use of LTL.

c**SUDS** - Software Understanding & Diagnosis System - A toolkit for testing, debugging, maintenance and understanding of software. Among its other abilities χSUDS can measure the test coverage achieved by the current test data set.

Cyclomatic Complexity Measurements - tools that attempt to measure the McCabe cyclomatic complexity, roughly the number of paths available. The idea being that the higher the measure the more likely it is to be coded wrong. There is a fair bit of empirical evidence that supports this notion.

Test Coverage Measurements - tools that count the number of code paths covered by the test data relative to all paths available, χSUDS is one such tool.

Aids - Tools used to identify critical functions of the system for incorporation into a fault tolerant system

Petri Nets - a petri net is a graphical representation of the system under consideration that focuses on the available states and rules for state transition. Once the net (graph) has been developed, undesirable states and interactions can be identified and corrected.

FMEA - Failure Modes and Effects Analysis - A cause and effect analysis of the ways in which a system fails are linked with the result of the failure.

FMEA was originally developed in the hardware world and has proved its worth many times. Software FMEA is most easily applied at the functional flow level or the data flow level. In both cases one assumes a failure, i.e., the software fails to satisfy its requirements, in the block under question and traces its effect throughout the system. This allows one to see the strengths and weaknesses of the current design with respect to the particular fault in question. Its appearance as a software design tool, or even as a way of thinking about software, is quite rare.

Fault Tree Analysis - A process for determining chain of failures necessary to reach a given undesirable event.

As with FMEA, Fault Tree Analysis was originally developed in the hardware world, where it has proved its worth as a design tool many times. To apply FTA to software one assumes a particular outcome and traces backward to determine the chain that would result in such an event. Unlike hardware, software is not immune to error propagation and so it is a chain that is identified rather than a number of separate failures. FTA is also quite rare in the software world.

Both FMEA and FTA can be used to identify critical paths that should be examined and tested with special rigor. FTA would seem to have applicability as a debug aid and, on those grounds alone, be more widely used.

Techniques - Structures or concepts used to implement the above methods

Assertions - extracted from formal methods - a statement and test of a code invariant, i.e. a statement about the code about to be executed that must be true.

Assertions have escaped the world of formal methods and are being more widely used every day. The primary reason seems to be that developers not versed in formal methods can, with little effort, understand the idea of an assertion and successfully make use of them. They can be applied at several levels from asserting every line of code before it is executed to only asserting the entry conditions for major blocks. The most powerful use is translating the requirements to assertions and properly placing them in the code. This provides early detection of invalid requirements and, if maintained, automatic regression testing for the life of the system. The use of assertions can be a very powerful technique of identifying when a software system has gone astray.

Try Blocks/Retry Blocks - a procedure that takes inputs, processes them, and generates outputs / a retry block runs a data re-expression algorithm prior to re-processing.

Scenario Based Tests - a testing that simulates the operational uses of the software product. The aim is not only to verify the software but reveal operational states that the requirements fail to address. The Space Shuttle software development process makes extensive use of scenario based tests.

Voting - used in a multiprocess system - the outputs of each process are compared and the majority (or largest minority) are taken as the correct output.

Heartbeat - monitoring of a processor by checking the traffic in and out or by checking its response to commands. Differs from a watchdog in that the monitoring is much more frequent.

Watchdog - used in multiprocessor systems - one processor has the responsibility of monitoring the others. Generally, a watchdog looks for a response within some time limit as opposed to a heartbeat which looks for a response every cycle.

Checkpointing - capturing the data prior to execution of a process - this data is then used in another try or retry block.

Data Re-expression - an algorithm that alters data prior to processing to avoid mathematical errors (division by zero, etc.)

Symbolic execution - a verification technique that focuses on the logical relationship between inputs and outputs.

Ad Hoc Techniques - Techniques that make use of special knowledge about the system and its environment in question. The approaches will generally be unique to one particular system.

System consistency checks - Logical relationships are established for very high level system outputs, if possible. The system continually checks to ensure the relationship maintains its correct state.

Statistical analysis of outputs - a technique that makes use of the fact that nature is generally not discontinuous. A discontinuity in outputs would be indicative of system misbehavior.

Graceful degradation - process of reducing demands on the processors by gradually reducing system responsibility to critical systems. This is not so much a method of providing fault tolerance as a response to an identified system fault.

"Safe corridor" - basic idea is to predict an acceptable path from the current location to the destination and pronounce the system faulty if it fails to stay within the corridor.

Table 1 is a summary of the methods, tools, and techniques discussed as well as their realm of applicability.

Table 1

	Applicable Stage						
	Rqmts Spec.	Design	Code	Testing	Post Production	Detect Erroneous Implement ation	
N-Version	·	·	·	·		·	
Recovery Block	·	·	·	·			
Distributed Recovery Block	·	·	·	·			
Try Block		·	·				
Scenario Test	·	·	·			·	
Voting	·	·	·	·			
Heartbeat	·	·	·	·			
Watchdog	·	·	·	·			
Checkpoint		·	·				
Data Re-expression		·	·				
Cyclomatic Complexity Measurements			·				
Test Coverage Measurements				·	·		
Symbolic Execution					·	·	
Petri Nets				·	·	·	
Fault Tree Analysis		·	·				
Failure Modes and Effects Analysis	·	·	·	·			
Formal Methods	·				·	·	
SPIN (Model State Checking)					·	·	
Assertions	·			·		·	
ReSoFT		·	·	·			
CORBA		·	·				

Appendix A

Resources Consulted

Internet Links to Related Documents

http://fermi.sohar.com/publications/ftcomp.html - Fault Tolerant Computing
http://www.computer.org/cspress/catalog/pr07146.htm - 25th Int Symp on Fault Tolerant Comp
http://www.informatik.hu-berlin.de/~rok/Lectures/FTC/ftc.html - Prof. Malek: Fault Tol Comp
http://casaturn.kaist.ac.kr/~sikang/course/CS714-96/ CS714: Fault Tolerant Systems Readings
http://www.ftcs.org/index.html - FTCS29
http://www.computer.org/conferen/proceed/ftcs95/ftcs95tc.htm -FTCS 95
http://catless.ncl.ac.uk/Risks/
http://www.rvs.uni-bielefeld.de/~ladkin/Incidents/FBW.html
http://www.rvs.uni-bielefeld.de/~ladkin/Reports/risk.html
http://munday.jsc.nasa.gov/
http://www.cs.washington.edu/homes/leveson/
http://www.wvu.edu/users/research/www/techbriefs/institutetechbrief.html
http://www.sei.cmu.edu/sei-home.html
http://www.law.vill.edu/Fed-Agency/fedwebloc.html
http://www.laas.research.ec.org/esp-syn/text/ec-us051.html
http://www.laas.research.ec.org/pdcs/ppr1994/exec-summary/task2.html
http://www.soften.ktu.lt/en/jep-06032/city/courses/IFPR402/reliab.html
http://newcastle.cabernet.esprit.ec.org/pdcs/ppr1993/state-of-art/task2.html
http://www.newcastle.research.ec.org/esp-syn/text/6362.html
http://www.dcs.warwick.ac.uk/~mathai/publications.html
http://people.ne.mediaone.net/edj/index.html
http://newcastle.cabernet.esprit.ec.org/pdcs/ppr1993/work-summary/task2.html
http://www.newcastle.research.ec.org/cabernet/workshops/3rd-plenary-papers/25-arlat.html
http://www3.informatik.uni-erlangen.de:1200/Staff/balbach/verify/dcca_fm2html.doc.html
http://bonda.cnuce.cnr.it/Documentation/Reports/abstract96/Nelli-C96-09
http://www.aosi.com/yeager/SoftwareEngineering.html
http://newcastle.cabernet.esprit.ec.org/pdcs/ppr1993/state-of-art/task4.html
http://www-dse.doc.ic.ac.uk/~nd/surprise_95/journal/vol2/eaj2/article2.html
http://www.comlab.ox.ac.uk/archive/safety.html
http://www.pathfinder.com/corp/tech/papers.html
http://www.iist.unu.edu/~tj/semlog/semlog_2.html
http://composer.ecn.purdue.edu/~fuchs/fuchs/fuchspub.html
http://www.cs.uiuc.edu/research/areaindex.html
http://www.dcs.shef.ac.uk/~u6pjw2/formal.html
http://www.laas.research.ec.org/cabernet/workshops/3rd-plenary-papers/30-feldt.html
http://www.eet.com/news/98/994news/realtime.html
http://www.cs.pitt.edu/~egan/FORTS/SantaFepres.html
http://www.cs.umr.edu/ecl.html

Avizienis, A. "The Methodology of N-Version Programming." Ch. 2 In *Software Fault Tolerance*, Lyu, M., Ed. (John Wiley & Sons), 1995.

Dugan, J. B., and M.R. Lyu. "Dependability Modeling for Fault-Tolerant Software and Systems." Ch. 5 In *Software Fault Tolerance*, Lyu, M., Ed. (John Wiley & Sons), 1995.

Pullum, L. L. "Data Diverse Software Fault Tolerance Techniques for C3I Technologies: Phase I Final Report." Quality Research Associates Technical Report, RL-TR-95-15, Rome Laboratory Contract F30602-94-C-0174, Feb., 1995.

Pullum, L. L. "Tutorial: Software Fault Tolerance ." *International Symposium on Software Reliability Engineering*, 1997-8.

Pullum, L. L. "Tutorial: Software Fault Tolerance ." *Reliability and Maintainability Symposium*, 1999.

Continuing Computer Failures. *IEEE Computer*, v.23, no. 7. July 1990.

Meyer, J. F., and R. D. Schlichting eds. "Dependable Computing for Critical Applications 2." Springer-Verlag Wien, New York, 1992.

Lyu, Michael R., and Algirdas Avizienis. *Assuring Design Diversity in N-Version Software: A design paradigm for N-Version programming.*

Nicola, Victor F., and Ambuj Goyal. *Limits of Parallelism in Fault-Tolerant Multiprocessors*.

Software System Safety:

Leveson, N.G. Safeware: System Safety in the Computer Age,Addison-WesleyPublishing Company, 1995

Leveson, N.G. and P.R. Harvey.``Analyzing Software Safety," IEEE Transactions on SoftwareEngineering, vol. SE-9, no. 5, 1983

Leveson, N.G. and Stolzy,J.L. ``Safety Analysis Using Petri Nets," IEEE Trans. on SoftwareEngineering, Vo l. SE-13, No. 3, March 1987, pp. 386-397.

Leveson, N.G. ``Software Safety: Why, What, and How," ACMComputing Surveys, Vo l. 18, No. 2, June 1986, pp. 125-163

Leveson, N.G. ``Software Safety in Embedded Computer Systems," Communications of the ACM, February,1991

Jaffe, M.S., Leveson, N.G., Heimdahl, M., and Melhart, B. ``Software Requirements Analysis for Real-Time Process-Control Software," IEEE Trans. on SoftwareEngineering, March 1991.

Leveson, N.G., Cha, S.S., Shimeall, T.J. ``Safety Verification of Ada Programs using Software Fault Trees," IEEE Software, July 1991

Leveson, N.G. and Turner,C.L. ``AnInvestigation of the Therac-25 Accidents," IEEE Computer, July 1993.

Leveson, N.G., Heimdahl, M.P.E., Hildreth, H., and Reese, J.D. ``Requirements Specification for Process-Control Systems," IEEE Transactions on SoftwareEngineering, September,1994.

Leveson, N.G. ``High-Pressure Steam Engines and Computer Software," IEEE Computer,October,1994.

Software Fault Tolerance

Knight, J.C. and Leveson, N.G. ``An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming," IEEE Transactions on Software Engineering, Vo l. SE-12, No. 1, January 1986, pp. 96-109

Brilliant, S.S., Knight, J.C., and Leveson, N.G. ``Analysis of Faults in an N-Version Software Experiment," IEEE Trans. on Software Engineering, Vol. SE-16, No. 2, February 1990.

Brilliant, S., Knight, J.C., and Leveson, N.G. ``The Consistent Comparison Problem in N-Version Programming," IEEE Trans. on Software Engineering, Vol. SE-15, No. 11, November 1989.

Leveson, N.G., Cha, S.S., Knight, J.C., and Shimeall, T.J. ``The Use of Self Checks and Voting in Software Error Detections: An Empirical Study,"IEEE Trans. on Software Engineering, Vol. SE-16, No. 4, April, 1990.

Shimeall, T., Leveson, N.G. ``An Empirical Comparison of Software Fault Tolerance and Fault Elimination," IEEE Trans. on Software Engineering, Vol. SE-17, No. 2, February 1991, pp. 173-183.

Knight, J.C. and Leveson, N.G., ``A Reply to the Criticisms of the Knight and Leveson Experiment," ACM Software Engineering Notes, January 1990.